**BÀI 2 - BIẾN VÀ BIỂU THỨC**

# Giá trị và kiểu giá trị

*Giá trị* là một trong những thứ cơ bản mà chương trình sử dụng, chẳng hạn như một chữ cái hoặc một số. Từ trước đến nay, chúng ta đã thấy các giá trị 1, 2 và " Hello, World!"

Các giá trị này thuộc các *kiểu* khác nhau: 2 là số nguyên và " Hello, World!" là một string (xâu), nó được gọi là string vì nó chứa một "xâu" các chữ cái. Bạn (và trình thông dịch) có thể xác định các string vì chúng được đặt trong dấu ngoặc kép.

Cũng có thể dùng câu lệnh print (in) cho các số nguyên. Chúng ta sử dụng lệnh python để khởi động trình thông dịch.



Nếu bạn không chắc kiểu của giá trị là gì, trình thông dịch có thể cho bạn biết.



Không có gì ngạc nhiên khi các string thuộc kiểu str và số nguyên thuộc kiểu int. Có những giá trị thì khó đoán kiểu hơn, các số có dấu thập phân thuộc kiểu float, bởi vì những số này được biểu diễn ở định dạng *dấu phẩy động*.



Còn các giá trị như “17” và “3.2” thì sao? Chúng trông giống như số, nhưng chúng nằm trong dấu ngoặc kép giống như string.



Chúng là string.

Khi bạn nhập một số nguyên lớn, cứ ba chữ số thì bạn nên tách chúng ra bởi dấu phẩy, như 1,000,000. Đây không phải là một số nguyên hợp lệ trong Python, nhưng nó hợp lệ:



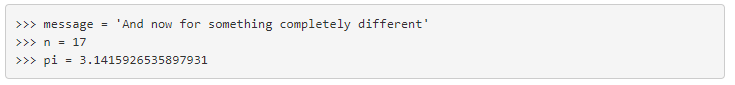
Chà, đây không phải là những gì chúng tôi mong đợi! Python hiểu 1,000,000 là một chuỗi các số nguyên được phân tách bằng dấu phẩy, nó sẽ in số ra với khoảng trắng ở giữa.

Đây là ví dụ đầu tiên mà chúng ta thấy về lỗi ngữ nghĩa: code chạy mà không có thông báo lỗi, nhưng nó không thực hiện đúng ý chúng ta.

# Biến

Một trong những tính năng mạnh mẽ nhất của ngôn ngữ lập trình là khả năng thao tác với *các biến*. Biến là một tên tham chiếu đến một giá trị.

Một *câu lệnh gán* tạo các biến mới và cung cấp cho chúng các giá trị:



Ví dụ này thực hiện 3 phép gán. Phép gán đầu tiên gán một string cho một biến mới có tên là message; phép gán thứ hai gán số nguyên 17 cho n; phép gán thứ ba gán giá trị (gần đúng) của π cho pi.

Để hiển thị giá trị của một biến, bạn có thể sử dụng câu lệnh print:



Kiểu của một biến là kiểu của giá trị mà nó tham chiếutới.



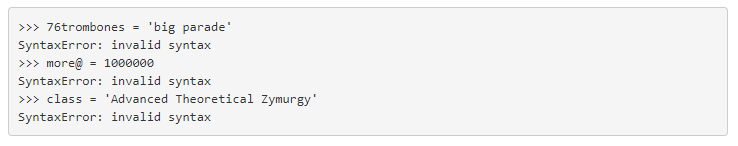
# Tên biến và từ khóa

Thường thì, các lập trình viên chọn tên cho các biến sao cho có ý nghĩa và biểu thị biến đó được sử dụng để làm gì.

Tên biến có thể dài tùy ý. Chúng có thể chứa cả chữ cái và số, nhưng chúng không thể bắt đầu bằng một số. Sử dụng chữ hoa là hợp lệ, nhưng bạn nên bắt đầu tên biến bằng chữ thường (sau này, bạn sẽ biết lý do).

Có thể dùng ký tự gạch dưới (\_) trong tên. Nó thường được sử dụng trong các tên có nhiều từ, chẳng hạn như my\_name hoặc airspeed\_of\_unladen\_swallow. Tên biến có thể bắt đầu bằng ký tự gạch dưới, nhưng thường thì, chúng ta sẽ không làm điều này trừ khi chúng tôi đang viết code thư viện cho người khác sử dụng.

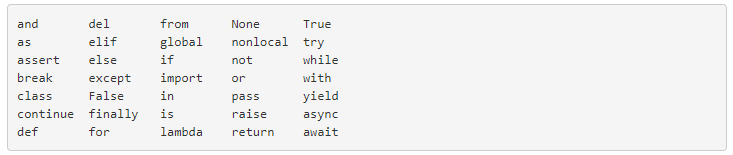
Nếu bạn đặt tên không hợp lệ cho một biến, bạn sẽ gặp lỗi cú pháp:



**76trombones** thì không hợp lệ vì nó bắt đầu bằng một số. **more@** thì không hợp lệ vì nó chứa một ký tự không hợp lệ, @. Nhưng còn **class** thì sao?

Hóa ra rằng **class** là một *từ khóa* của Python. Trình thông dịch sử dụng các từ khóa để nhận ra cấu trúc của chương trình và chúng ta không thể dùng chúng làm tên biến.

Python có 35 từ khóa dành riêng:



Bạn nên làm cách nào đó để nhớ danh sách này. Nếu trình thông dịch phàn nàn về một tên biến của bạn và bạn không biết lý do tại sao, hãy xem liệu nó có trong danh sách này không.

# Câu lệnh

*Câu lệnh* là một đơn vị code mà trình thông dịch Python có thể thực thi. Chúng ta đã thấy hai loại câu lệnh: print là một câu lệnh biểu thức và phép gán.

Khi bạn nhập một câu lệnh ở chế độ tương tác, trình thông dịch thực thi câu lệnh đó và hiển thị kết quả, nếu có.

Một tập lệnh thường chứa một chuỗi các câu lệnh. Nếu có nhiều hơn một câu lệnh, kết quả sẽ xuất hiện lần lượt khi các câu lệnh thực thi.

Ví dụ, tập lệnh



cho output (đầu ra) sau



Câu lệnh gán không tạo ra ouput

# Toán tử và toán hạng

*Toán tử* là các ký hiệu đặc biệt đại diện cho các phép tính như cộng và nhân. Toán tử được áp dụng cho giá trị nào thì giá trị đó được gọi là toán hạng.

Các toán tử +, -, \*, /, và \*\* thực hiện các phép tính cộng, trừ, nhân, chia và lũy thừa, như trong các ví dụ sau:



Đã có một sự thay đổi trong toán tử chia giữa Python 2.x và Python 3.x. Trong Python 3.x, kết quả của phép chia này là kết quả dấu phẩy động:



Toán tử chia trong Python 2.0 sẽ chia hai số nguyên và cắt ngắn kết quả thành một số nguyên:



Để có được câu trả lời tương tự trong Python 3.0, hãy sử dụng phép chia làm tròn xuống (// số nguyên).



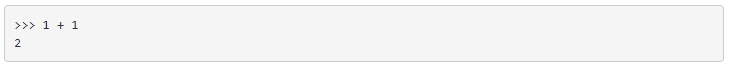
Trong Python 3.0, các phép chia số nguyên thực hiện nhiều thứ hơn so với nhập biểu thức trên máy tính bỏ túi.

# Biểu thức

Một *biểu thức* là sự kết hợp của các giá trị, biến và toán tử. Một giá trị ở một mình được coi là một biểu thức và một biến cũng vậy, vì vậy tất cả các biểu thức sau đều là biểu thức hợp lệ (giả sử rằng biến x đã được gán một giá trị):



Nếu bạn nhập một biểu thức trong chế độ tương tác, trình thông dịch sẽ *tính* biểu thức đó và hiển thị kết quả:



Nhưng trong một tập lệnh, một biểu thức ở một mình thì cũng không có tác dụng gì! Đây là một nguyên nhân phổ biến khiến cho người mới bắt đầu dễ nhầm lẫn.

# Thứ tự phép toán

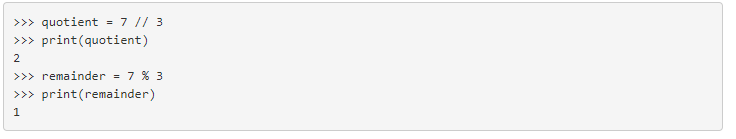
Khi nhiều toán tử xuất hiện trong một biểu thức, thứ tự tính toán phụ thuộc vào các *quy tắc ưu tiên*. Đối với các toán tử toán học, Python tuân theo quy ước toán học. Từ viết tắt PEMDAS là một cách hữu ích để ghi nhớ các quy tắc:

* Dấu ngoặc đơn có mức độ ưu tiên cao nhất và có thể được sử dụng để buộc một biểu thức tính toán theo thứ tự bạn muốn. Vì các biểu thức trong dấu ngoặc đơn được tính đầu tiên nên 2 \* (3-1) là 4 và (1+1)\*\*(5-2) là 8. Bạn cũng có thể sử dụng dấu ngoặc đơn để khiến một biểu thức dễ đọc hơn, như trong (minute \* 100) / 60, dù là làm như vậy thì kết quả cũng không thay đổi.
* Luỹ thừa có mức độ ưu tiên cao thứ 2, vì vậy 2\*\*1+1 là 3, không phải 4 và 3\*1\*\*3 là 3, không phải 27.
* Phép nhân và phép chia có cùng mức ưu tiên, cao hơn phép cộng và phép trừ, (phép cộng và phép trừ cũng có cùng mức độ ưu tiên). Vì vậy, 2\*3-1 là 5, không phải 4 và 6+4/2 là 8, không phải 5.
* Các toán tử có cùng thứ tự ưu tiên được tính theo thứ tự từ trái sang phải. Vì vậy, biểu thức 5-3-1 là 1, không phải 3, bởi vì 5-3 được tính trước và sau đó lấy 2 trừ đi 1.

Khi không chắc chắn, hãy luôn đặt dấu ngoặc đơn trong biểu thức của bạn để đảm bảo các phép tính được thực hiện theo thứ tự bạn muốn.

# Toán tử Modulo

*Toán tử modulo* hoạt động trên các số nguyên và tính số dư khi toán hạng đầu tiên chia cho toán hạng thứ hai. Trong Python, toán tử modulo là dấu phần trăm (%). Cú pháp giống như các toán tử khác:



Vậy 7 chia 3 được 2 còn dư 1.

Hóa ra, toán tử modulo lại rất hữu ích. Ví dụ, bạn có thể kiểm tra xem một số có chia hết cho số khác hay không: nếu x % y bằng 0, thì x chia hết cho y.

Bạn cũng có thể trích xuất chữ số ngoài cùng bên phải hoặc các chữ số từ một số. Ví dụ: x % 10 cho kết quả là chữ số tận cùng bên phải của x (trong cơ số 10). Tương tự, , x % 100 cho kết quả là hai chữ số cuối cùng.

# Các phép toán với string

Toán tử + hoạt động với string, nhưng nó không phải là phép cộng theo nghĩa toán học. Thay vào đó, nó thực hiện phép nối, có nghĩa là liên kết phần cuối string nó với phần đầu string kia để nối chúng lại với nhau. Ví dụ:



Toán tử \* cũng hoạt động với các string bằng cách nhân nội dung của một string với một số nguyên. Ví dụ:

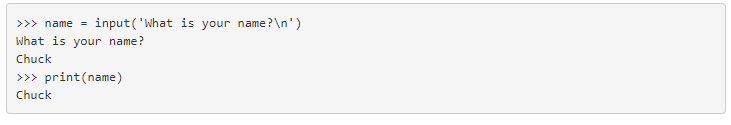


# Yêu cầu người dùng nhập liệu

Đôi khi, chúng ta muốn người dùng nhập giá trị của một biến. Python cung cấp một hàm tích hợp được gọi là input lấy đầu vào từ bàn phím. Khi hàm này được gọi, chương trình sẽ dừng lại và đợi người dùng gõ một cái gì đó. Khi người dùng nhấn Return hoặc Enter, chương trình sẽ tiếp tục và hàm input trả về những gì người dùng đã nhập dưới dạng một string.

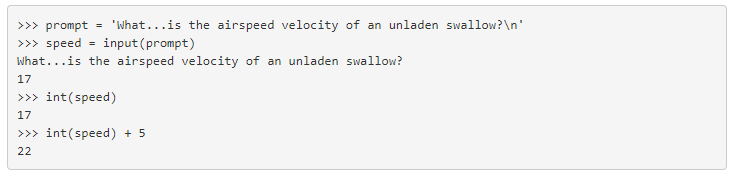


Trước khi nhận được input từ người dùng, bạn nên in một lời nhắc cho người dùng biết họ cần nhập gì. Bạn có thể truyền một string vào input để hiển thị cho người dùng trước khi chương trình tạm dừng để chờ người dùng nhập:

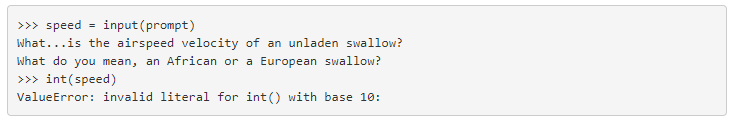


Chuỗi \n cuối của lời nhắc đại diện cho *một dòng mới*, là một ký tự đặc biệt để ngắt dòng. Đó là lý do tại sao input của người dùng xuất hiện bên dưới lời nhắc.

Nếu bạn muốn người dùng nhập một số nguyên, bạn có thể thử chuyển đổi giá trị trả về thành int bằng cách sử dụng hàm int() :



Nhưng nếu người dùng không nhập một string các chữ số mà nhập thứ khác, bạn sẽ gặp lỗi:



Chúng ta sẽ xem cách xử lý loại lỗi này ở phần sau.

# Comment (chú thích)

Khi các chương trình ngày càng lớn hơn và phức tạp hơn thì sẽ khó đọc chúng hơn. Các ngôn ngữ chính thức có mật độ dày đặc và thường rất khó để nhìn vào một đoạn code và biết được nó đang làm gì hoặc tại sao.

Vì lý do này, bạn nên thêm comment vào chương trình để giải thích chương trình đang làm gì bằng ngôn ngữ tự nhiên của con người. Chúng được gọi là comment và trong Python, chúng bắt đầu bằng ký hiệu #:



Trong trường hợp này, comment tự xuất hiện trên một dòng. Bạn cũng có thể đặt comment ở cuối dòng:



Mọi thứ từ dấu # đến cuối dòng đều bị bỏ qua; nó không ảnh hưởng đến chương trình.

Comment hữu ích nhất khi chúng dùng nó để giải thích các tính năng không rõ ràng của code. Chúng ta có thể giả định rằng người đọc có thể biết code *làm gì*; vậy, nên dùng comment để giải thích *lý do tại sao*.

Comment này là thừa và vô ích:



Comment này chứa thông tin hữu ích không có trong code:



Nếu có tên biến tốt thì không cần comment, nhưng nếu tên dài thì có thể làm cho các biểu thức phức tạp trở nên khó đọc, vì vậy, đôi khi, chúng ta cần phải đánh đổi.

1. **Chọn tên biến dễ nhớ**

Miễn là bạn tuân theo các quy tắc đơn giản khi đặt tên biến và tránh các từ dành riêng, bạn có rất nhiều lựa chọn để đặt tên cho biến. Lúc đầu, sự lựa chọn này có thể gây nhầm lẫn khi bạn đọc một chương trình và khi bạn viết các chương trình. Ví dụ: mục tiêu của ba chương trình sau đây giống hệt nhau, nhưng khi bạn đọc chúng và cố gắng hiểu chúng thì chúng lại rất khác.

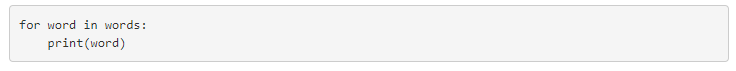


Đối với trình thông dịch Python, cả ba chương trình này hoàn toàn giống nhau nhưng con người lại thấy và hiểu các chương trình này hoàn toàn khác nhau. Con người sẽ nhanh chóng hiểu được ý định của chương trình thứ hai vì lập trình viên đã chọn các tên biến phản ánh ý định của họ về dữ liệu sẽ được lưu trữ trong mỗi biến.

Chúng ta gọi những tên biến được lựa chọn một cách khôn ngoan này là “*tên biến dễ nhớ*” (mnemonic). Từ mnemonic có nghĩa là "hỗ trợ trí nhớ". Chúng ta chọn tên biến dễ nhớ để giúp chúng ta nhớ lý do tại sao chúng ta tạo biến.

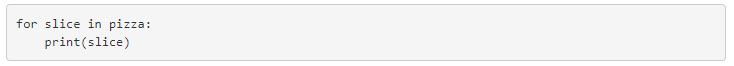
Mặc dù những điều này nghe thật tuyệt vời và bạn nên sử dụng tên biến dễ nhớ, nhưng tên biến dễ nhớ có thể cản trở khả năng phân tích cú pháp và hiểu code của một lập trình viên mới bắt đầu. Điều này là do các lập trình viên mới bắt đầu chưa nhớ các từ dành riêng (chỉ có 33 từ) và đôi khi các biến có tên quá mô tả sẽ bắt đầu trông giống như một phần của ngôn ngữ lập trình chứ không chỉ là các tên biến được lựa chọn tốt.

Hãy xem nhanh code mẫu Python sau đây, nó lặp lại một số dữ liệu. Chúng ta sẽ sớm đề cập đến các vòng lặp, nhưng bây giờ hãy cố gắng giải đáp xem điều này có nghĩa là gì:



Có chuyện gì đang xảy ra ở đây? Mã thông báo (token) nào (for, word, in, v.v.) là các từ dành riêng và mã thông báo nào chỉ là tên biến? Python có hiểu khái niệm về từ ở cấp độ cơ bản không? Các lập trình viên mới bắt đầu gặp khó khăn khi phân biệt phần nào của code phải giống như ví dụ này và phần nào là do lập trình viên tự chọn.

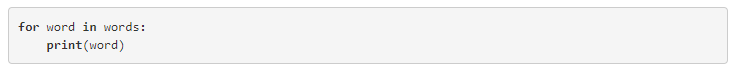
Code sau tương đương với code trên:



Lập trình viên mới bắt đầu sẽ dễ dàng nhìn vào đoạn code này và biết phần nào là các từ dành riêng do Python định nghĩa và phần nào là tên biến do người lập trình chọn. Rõ ràng là Python không có hiểu biết cơ bản về bánh pizza và các miếng pizza (slice) và nó cũng không biết một chiếc bánh pizza có nhiều miếng.

Nhưng nếu mục đích của chương trình thực sự là để đọc dữ liệu và tìm kiếm các từ trong dữ liệu, thì pizza và slice là những tên biến rất khó nhớ. Nếu chọn chúng làm tên biến thì sẽ làm phân tán ý nghĩa của chương trình.

Sau một khoảng thời gian ngắn, bạn sẽ biết những từ dành riêng phổ biến nhất và bạn sẽ bắt đầu thấy những từ dành riêng tự hiện rõ trước mắt:



Các phần của code do Python định nghĩa (**for, in, print, and**:) được in đậm và các biến do người lập trình chọn (word và words) không được in đậm. Nhiều trình soạn thảo văn bản biết về cú pháp Python và sẽ tô màu các từ dành riêng khác so với những từ khác để giúp bạn tách các biến và các từ dành riêng dễ dàng hơn. Sau một thời gian, bạn sẽ bắt đầu đọc Python và nhanh chóng xác định được đâu là biến và đâu là từ dành riêng.

1. **Gỡ lỗi**

Bây giờ, lỗi cú pháp mà bạn dễ mắc phải nhất là tên biến không hợp lệ, như **class và yield**, chúng là các từ khóa, hoặc odd~job và US$ chứa các ký tự không hợp lệ.

Nếu bạn đặt một khoảng trắng trong một tên biến, Python nghĩ rằng đó là hai toán hạng không có toán tử:



Đối với lỗi cú pháp, các thông báo lỗi không quá hữu ích. Các thông báo phổ biến nhất là **SyntaxError: invalid syntax và SyntaxError: invalid token**, cả hai đều không có nhiều thông tin.

Lỗi thời gian chạy dễ mắc phải nhất là "use before def;" nghĩa là cố gắng sử dụng một biến trước khi chỉ định một giá trị. Điều này có thể xảy ra nếu bạn đánh vần tên biến sai:



Tên biến phân biệt chữ hoa chữ thường, vì vậy LaTeX sẽ khác latex.

Tại thời điểm này, nguyên nhân dễ gây ra lỗi ngữ nghĩa nhất là thứ tự của các phép toán. Ví dụ: để tính 1/2*π*, bạn có thể sẽ muốn viết



Nhưng phép chia xảy ra trước, vì vậy bạn sẽ nhận được kết quả là *π/2*, thế thì sẽ khác với bên trên! Không có cách nào để Python biết bạn muốn viết gì, vì vậy trong trường hợp này, bạn không nhận được thông báo lỗi; bạn chỉ nhận được câu trả lời sai.